03

JDBC y sus problemas

Transcripción

[00:00] Hola. Para comenzar nuestra capacitación en JPA analizaremos la siguiente motivación. Para que se creó JPA y cuáles problemas resuelve. Además, discutiremos sobre Hibernate y otras implementaciones.

[00:12] En este video también cubriremos sobre JDBC, que es la tecnología predeterminada de Java para acceder a bases de datos relacionales. Aquellos que aprendieron a programar en Java, probablemente al desarrollar sistemas necesitarán acceder a bases de datos relacionales y para acceder a estas bases de datos con Java o por ejemplo SQL u Oracle la tecnología estándar utilizada es JDBC.

[00:35] Java nació en 1995 y en 1997 llegó JDBC. Antes de JDBC, si queríamos acceder a una base de datos era necesario aprender tecnologías que son bastante complejas y hacer todo manualmente.

[00:48] Había una conexión con la base de datos de hacer toda la comunicación usando el protocolo específico para esta base datos y esto ocasionaba bastante trabajo y era bastante complicado. JDBC vino a facilitar este proceso. JDBC no es más que una especificación para acceder a las bases de datos relacionales, por lo tanto solo una capa de abstracción para acceder desde el código, independientemente del protocolo.

[01:11] En otras palabras JDBC apareció como una capa para simplificar el acceso y facilitar los intercambios de bases de datos. A partir de esto ya no es necesario conocer el protocolo de MySQL o de Oracle y conocer los detalles

técnicos o seguir abriendo el socket y seguir haciendo la comunicación manual con la base de datos. Solo basta utilizar JDBC.

[01:32] Cualquiera que haya estudiado JDBC, aquí en Alura tenemos cursos sobre JDBC, sabe que necesitamos tener un controlador. Este controlador es un archivo jar que tiene las clases para conectar a la base de datos, es decir es una implementación de la base de datos sobre JDBC.

[01:49] Entonces si queremos acceder a MySQL, vamos a descargar el controlador de MySQl. Para cambiar a la base la base de datos a Postgre por ejemplo, vamos a cambiar este controlador que es un archivo jar, al controlador de Postgre. Ambos implementan JDBC, por lo que el impacto en el código es mínimo.

[02:09] Al pasar de una base de datos a otra, tenemos pocos cambios en el código ya que básicamente comimos la configuración, pero la mayor parte del código que se comunica con la base de datos sigue siendo la misma. Esto lo hace mucho más fácil porque no estamos atados a un único proveedor ni una única base de datos.

[02:28] Para mantener el código JDBC disperso en varios puntos de la aplicación se creó un patrón de diseño, que es el DAO, Data Access Object. Con este podemos aislar todo el pedido de JDBC dentro de una sola clase, para no quedarnos con los códigos connection result set dispersos por toda la aplicación que son clases de JDBC, que nos permiten el acceso a la base de datos.

[02:51] Básicamente tenemos una clase en la aplicación, un controlador y un servicio o algo de este tipo. Esta clase contiene la lógica de negocio, y en esta lógica es la que necesitamos para acceder a la base de datos. Es decir, no instanciamos ni llevamos clase Js, sino que vamos a llamar la clase DAO, influenciemos la clase DAO y en esta clase donde se abstrae y encapsula el código JDBC.

[03:15] También es el que hace el puente con la base de datos. Entonces hubo una división de responsabilidades en la aplicación. Mirando desde afuera la clase DAO habría algo como esto.

[01:27] Una clase RegistroDeProductoService y necesitaríamos una clase DAO, método registrar producto. Entonces recibiríamos el objeto producto, tendríamos también las reglas de negocio, una validación, cálculo y luego lo llamaríamos la clase dao.registrar.

[03:45] Mirando desde afuera no es posible saber cómo está funcionando esa clase DAO, si está usando JDBC, si la persistencia está en una base de datos, si está en un archivo, si está en memoria, en un servicio externo, tampoco si es MySQL u Oracle. El código es bastante fácil de usar, llamamos a la clase DAO, luego el método registrar pasamos el producto y eso es todo.

[04:06] No estamos presos con la forma en la que se implementó el método registrar. Así que no tenemos el acceso al API de JDBC, está se encuentra aislada. Por fuera el código se ve bastante limpia pero dentro de la clase DAO tenemos un problema. En las clases DAO continuamos usando JDBC, terminamos teniendo un código bastante complicado porque necesitamos usar la API de JDBC, una API bastante antigua ya que fue creada en 1997, con mucha verbosidad.

[04:37] Y eso generó que la gente desarrollara como cierta aversión a Java creyendo que Java es muy burocrático así como JDBC. Por lo tanto necesario tratar con clases como preparedStatement, connection, resultSet y también utilizar try catch ya que arroja excepciones comprobadas, además de ensamblar las consultas manuales y usar el preparedStatement para evitar problemas de inyección de SQL.

[05:04] Al final todo esto hace que el código sea bastante complicado. Ese tipo de código JDBC, a pesar de que funciona bien, tiene algunos inconvenientes que han hecho que la gente piense en otras alternativas. JDBC tienes dos

grandes problemas que han motivado surgimiento de tecnologías como Hibernate y JPA.

[05:24] El primer problema es que el código es demasiado detallado, por ejemplo para guardar un producto en la base datos necesitamos al menos unas 30 líneas de código. En un código tan grande el mantenimiento es bastante difícil y requiere mucho tiempo. A veces es necesario ensamblar una consulta nativa desde la base de datos y el código se vuelve cada vez más complicado y difícil de entender.

[05:47] Esto es un gran problema y no es el más difícil, no es el problema más grande de todos. El segundo problema es el alto acoplamiento con las bases de datos. Cuando trabajamos con JDBC, tenemos una acoplamiento muy grande con la base de datos.

[06:00] Significa que si nosotros por ejemplo, realizamos un cambio en la base de datos como el nombre de la tabla o de alguna columna cualquier otro elemento, vamos a impactar ampliamente dentro de la aplicación. Tendremos que cambiar en las clases DAO y el problema es que por ejemplo cuando cambiamos el nombre en la tabla productos puede ocurrir que además de existir en el producto DAO, hay otras clases del dado que realicen joins con la tabla producto.

[06:30] Es decir, pueden haber diferentes DAO utilizando la tabla producto y como nosotros hicimos una alteración en el nombre, tendríamos que buscar todos los elementos que hacen referencia a esa tabla producto y hacer este cambio de nombre.

[06:43] Esto nos lleva un alto acoplamiento en la base de datos. Cualquier cambio en un lado va a tener un alto impacto en el otro. Estos son los dos grandes problemas que la gente comenzó a notar con JDBC y debido a ello ha perfeccionado ideas sobre cómo reducir el impacto que generan estos problemas. De este proceso surgió JPA.

[07:00] En el siguiente video, discutiremos con más calma cómo se creó JPA y cómo resolvió los problemas de JDBC. Nos vemos en el siguiente video.